Title: **Mining Approximate Multi-Relational Patterns**

Authors: Eirini Spyropoulou; Tijl De Bie

The final publication is available at IEEE via https://doi.org/10.1109/DSAA.2014.7058115

Mining approximate multi-relational patterns

Eirini Spyropoulou
Toshiba Research Europe Limited
Bristol, UK
Email: eirini.spyropoulou@toshiba-trel.com

Tijl De Bie
Intelligent Systems Lab
University of Bristol
Bristol, UK
Email: tijl.debie@gmail.com

***Abstract*—Three recent trends aim to make local pattern mining more directly suited for use on data as it presents itself in practice, namely in a multi-relational form and affected by noise. The first of these trends is the generalisation of local pattern syntaxes to approximate, noise-tolerant, variants (notably fault-tolerant itemset mining and community detection). The second of these trends is to develop pattern syntaxes that are directly applicable to multi-relational data. The third one is to better quantify the interestingness of and redundancy between such local patterns.**

In this paper we leverage recent results from these lines of research to introduce a noise-tolerant pattern syntax for multi-relational data. We show how enumerating all patterns of this syntax in a given database can be done remarkably efficiently. We contribute a way to quantify the interestingness of these patterns, thus overcoming the pattern explosion problem. And finally, we show the usefulness of the pattern syntax and the scalability of the algorithm by presenting experimental results on real world and synthetic data.

I. Introduction

From frequent itemset mining to multi-relational pattern mining: Since the introduction of itemset mining, pattern mining research has mostly focused on algorithms for mining single relations. However, the wealth of datasets that are nowadays available are usually inherently multi-relational, comprising relations that are present between entities of different types. Therefore mining patterns that capture the complexity of such data is of great importance.

Think for example of a dataset containing entities of three different types: customers, customer characteristics, and products that they may buy. A pattern containing customers who have all bought the same products and share the same characteristics can be a lot more useful than one containing just customers and the products that they buy. Indeed, it may provide context and explanation for the patterns found: if a group of users all buying a given set of products shares a set of characteristics, these characteristics may well explain their buying behaviour. Moreover, the use of different data sources may increase the statistical power of the pattern mining method: a pattern involving a small set of users buying a small set of products may appear to be a matter of chance, but it can become statistically significant if these customers also share a set of characteristics.

To address this need, we recently proposed the pattern syntax of Maximal Complete Connected Subsets (MCCSs) [13], [14], [15], which enables mining such kind of patterns from multi-relational data. MCCSs are sets of entities of potentially different types that are related in all ways allowed by the database relations.

From exact to approximate patterns: The definition of an MCCS pattern implies that if a certain customer does not buy a certain product, they can never be part of the same MCCS with other customers who do not buy that product but who otherwise have identical buying behaviour and characteristics to that customer. Clearly, this makes this *exact* pattern syntax very strict and quite vulnerable to noise.

Vulnerability to noise is a property shared by many pattern syntaxes defined in an exact manner. Most notably this is the case for frequent itemsets, where a single missing item in a transaction may render the itemset infrequent. Thus, significant efforts have been expended to generalise the notion of the frequency of an itemset to a less strict notion, allowing some items to be missing from some of the transactions considered to support it [18], [8], [11], [2]. Another example is community detection in networks, for which *exact* clique patterns tend to be too restrictive and instead *approximate* clique patterns are sought, which are loosely defined as sets of nodes that are *densely* connected rather than *completely* [7], [17], [16], [6], [12].

For most local pattern types, the number of degrees of freedom in relaxing the exact nature of the patterns to something approximate is large. The challenge is to make the right trade-off between usability and resilience to noise on the one hand, and computational tractability on the other.

Approximate multi-relational patterns: The goal of this paper is to introduce an approximate multi-relational pattern type which we call α-CCSs, generalising the recently introduced MCCS pattern syntax. This is the first attempt at defining an approximate pattern syntax for this context. To do this, we will use a strategy similar to the one of Palla et al. [12] where communities in networks are defined as the union of cliques that overlap in at least a number of nodes.

Informally speaking, we will define α-CCSs as unions of MCCSs that overlap in a set of entities. This definition is highly resilient to noise, while the set of all such patterns can be enumerated remarkably efficiently (essentially as efficiently as enumerating all exact MCCSs).

Example: In the toy multi-relational data of Fig. 1, the set $\{$u1, u2, f1, f2, f3, a1$\}$ corresponds to an MCCS and the set $\{$u1, u2, u3, u4, f1, f2, a1, a3, a5$\}$ corresponds to another MCCS. These MCCSs share two films, two users and one actor. Their union, $\{$u1, u2, u3, u4, f1, f2, f3, a1, a3, a5$\}$, is

This work was done while Eirini Spyropoulou was a PhD student at the University of Bristol.

an approximate multi-relational pattern as we consider it in this paper.

films

users	f1	f2	f3	f4
u1	1	1	1	0
u2	1	1	1	0
u3	1	1	0	0
u4	1	1	0	1
u5	0	0	1	1

actors

films	a1	a2	a3	a4	a5
f1	1	0	1	0	1
f2	1	0	1	0	1
f3	1	0	0	1	0
f4	0	1	0	1	0

Fig. 1. Example of multi-relational data about films. There are three entity types namely users, films and actors, and two relationships. One between users and films specifying whether a user liked a film and one between films and actors specifying which actors played in which films.

Contributions: In this paper we propose the concept of an α-CCS in a multi-relational database, *an intuitive new pattern syntax* that is both *widely applicable and generic*, in being designed for multi-relational data, and *highly noise-resilient*, in tolerating a possibly large number of entity combinations that are covered by the pattern but are not related in the data. Remarkably, and in stark contrast with other approximate local pattern mining methods, this noise reslience comes *at virtually no extra computational cost*. Indeed, we will show that all α-CCSs can be enumerated by traversing the same search space as the one traversed for enumerating all exact MCCSs [15]. Finally, as the number of α-CCSs in a database is potentially large, we also propose a well-principled interestingness measure, allowing one to rank the results and in this way *overcome the pattern explosion problem*.

II. Multi-relational data and patterns

In Sec. II-A below we will recapitulate some key concepts introduced in our prior work [15], and that are necessary for the current paper. Then in Sec. II-B we formally define the new pattern syntax and prove a number of propositions which will allow the derivation of an efficient algorithm to mine all such patterns. But let us first start with some notation.

We formalize a **relational database** as a tuple $\mathbb{D} = (E, \mathbf{t}, \mathcal{R}, \mathbf{R})$ where E is a finite set of *entities* that is partitioned into k *entity types* by a mapping $\mathbf{t} : E \to \{1, \ldots, k\}$, i.e., $E = E_1 \dot\cup \ldots \dot\cup E_k$ with $E_i = \{e \in E \mid \mathbf{t}(e) = i\}$. Moreover, $\mathbf{R} \subseteq \{\{i, j\} \mid i, j \in \{1, \ldots, k\}, i \neq j\}$ is a set of *relationship types* such that for each $\{i, j\} \in \mathbf{R}$ there is a *binary relationship* $\mathcal{R}_{\{i,j\}} \subseteq \{\{e, e'\} \mid e \in E_i, e' \in E_j\}$. The set $\mathcal{R}$ then is the union of all these relationships, i.e., $\mathcal{R} = \bigcup_{\{i,j\} \in \mathbf{R}} \mathcal{R}_{\{i,j\}}$. We say that a set $\{e, e'\} \in \mathcal{R}$ is a *relationship instance*.

For convenience we will abuse notation and overload the operator $\mathbf{t}$ to be applicable also to a *set* of entities, yielding the *set* of their entity types.

A. The pattern syntax of CCSs and closed-CCSs

The pattern syntax of **Complete Connected Subsets (CCSs)** is defined based on the following definitions of completeness and connectedness [15].

Definition 1: A set $F \subseteq E$ is **complete** iff for all $e, e' \in F$ with $\{t(e), t(e')\} \in \mathbf{R}$ it holds that $\{e, e'\} \in \mathcal{R}_{\{t(e),t(e')\}}$.

Definition 2: A set $F \subseteq E$ is **connected** iff for all $e, e' \in F$ there is a sequence $e = e_1, \ldots, e_l = e'$ with $\{e_1, \ldots, e_l\} \subseteq F$ such that for $i \in \{1, \ldots, l-1\}$ it holds that $\{e_i, e_{i+1}\} \in \mathcal{R}$.

Using these notions, a CCS $F \subseteq E$ is defined as a set of entities that is both complete and connected. The set of all CCSs forms a so-called *set system* over the ground set E, further denoted as $\mathcal{F}_\mathbb{D}$ and formally defined as follows:

Definition 3: For a database $\mathbb{D} = (E, \mathbf{t}, \mathcal{R}, \mathbf{R})$ the **set system of CCSs**, is defined as

$$\mathcal{F}_\mathbb{D} = \{F \subseteq E \mid F \text{ connected } \wedge F \text{ complete}\}.$$

A **Maximal CCS (MCCS)** is a CCS to which no entity can be added without violating completeness or connectedness. We argued in [15] that MCCSs are of particular interest, and we derived an algorithm for enumerating them efficiently. This algorithm in fact enumerates another subset of all CCSs, namely the so-called *closed CCSs*, which form a relatively tight superset of all MCCSs. Enumerating closed CCSs will also form a central part of the algorithm for mining the newly proposed pattern syntax of α-CCSs (Sec. II-B). Therefore, it will prove helpful to discuss them in some detail here.

A **closed CCS** is defined as a fixed point under a *closure operator* ρ on the set system $\mathcal{F}_\mathbb{D}$, i.e. a function $\rho\colon \mathcal{F}_\mathbb{D} \to \mathcal{F}_\mathbb{D}$ with the following properties: (a) Extensivity: $F \subseteq \rho(F)$ for all $F \in \mathcal{F}_\mathbb{D}$; (b) Monotonicity: $\rho(F) \subseteq \rho(F')$ for all $F, F' \in \mathcal{F}_\mathbb{D}$ with $F \subseteq F'$; (c) Idempotence: $\rho(\rho(F)) = \rho(F)$ for all $F \in \mathcal{F}_\mathbb{D}$. To introduce the particular closure operator proposed in [15], two other operators need to be introduced first, $Aug\colon \mathcal{F}_\mathbb{D} \to \mathcal{P}(E)$ and $Comp\colon \mathcal{F}_\mathbb{D} \to \mathcal{P}(E)$.

For any CCS $F \in \mathcal{F}_\mathbb{D}$, the set $Aug(F)$ contains all elements $e \in E$ such that also $F \cup \{e\}$ is complete and connected and thus a CCS. Formally: $Aug(F) = \{e \in E \mid F \cup \{e\} \in \mathcal{F}_\mathbb{D}\}$.

For any $F \in \mathcal{F}_\mathbb{D}$ the set $Comp(F)$ is defined as the set of entities $e \in E$ such that $F \cup \{e\}$ is complete. For short we will refer to $Comp(F)$ as the *set of compatible entities* of F.

The closure operator used in [15] can conveniently be expressed in terms of the Aug and $Comp$ operators:

Definition 4: For a relational database $\mathbb{D} = (E, \mathbf{t}, \mathcal{R}, \mathbf{R})$, the operator $g : \mathcal{F}_\mathbb{D} \to \mathcal{P}(E)$ is defined as

$$g(F) = \{e \in Aug(F) \mid Comp(F \cup \{e\}) = Comp(F)\} \ .$$

The rationale behind the design of this operator and the proof that it is indeed a closure operator can be found in [15].

We conclude this subsection by highlighting a number of useful properties of the Aug and $Comp$ operators. It is immediately clear from their definitions that $F \subseteq Aug(F) \subseteq Comp(F)$. Further, the operator $Comp$ is anti-monotone, i.e., for $F, F' \in \mathcal{F}_\mathbb{D}$, $F \supseteq F'$, $Comp(F) \subseteq Comp(F')$.

B. The pattern syntax of α-CCSs

In [15] we introduced RMiner, an algorithm which enumerates all closed CCSs. The search space of RMiner consists of all closed CCSs, starting from the empty set, such that the children of every closed CCS, F, correspond to $g(F \cup e)$ for every $e \in Aug(F)$. The leaves of the search space correspond to MCCSs. Therefore, each closed CCS gives rise to several MCCSs that are supersets of it. These MCCSs share the entities in the closed CCS they are generated from as a common core.

The kind of pattern we consider in this paper is the union of the set of all MCCSs that can be found by extending a given closed CCS. More precisely, let us denote the set of closed CCSs as $\mathcal{C}$ and the set of MCCSs as $\mathcal{M}$. We define an **α-CCSs** pattern as any subset from E in the image of the operator $\alpha : \mathcal{C} \rightarrow \mathcal{P}(E)$, defined as follows:

Definition 5: Given a set $F \in \mathcal{C}$ we define $\alpha : \mathcal{C} \rightarrow \mathcal{P}(E)$ as

$$\alpha(F) = \bigcup_{\substack{\forall F' \supseteq F, \\ F' \in \mathcal{M}}} F' \ .$$

This pattern syntax is noise tolerant, in the sense that the union of a set of MCCSs extending a given closed CCS will not be a CCS itself. Yet, it will be coherent in that it has a closed CCS core. Moreover, we will show an efficient algorithm for enumerating all such patterns can be derived. This algorithm exploits the following three properties.

Proposition 1: For any $F \in \mathcal{C}$ it holds that $\alpha(F) \subseteq Comp(F)$.

Proof: Let us assume that $\alpha(F) \not\subseteq Comp(F)$. This means that there is an $e \in \alpha(F)$ such that $e \notin Comp(F)$. Since $e \in \alpha(F)$, there is an $F' \supseteq F$, $F' \in \mathcal{M}$ such that $e \in F'$. Since $e \notin Comp(F)$, there is an $e' \in F$ such that $\{\mathbf{t}(e), \mathbf{t}(e')\} \in \mathbf{R}$ and $\{e, e'\} \notin \mathcal{R}$. However, from $e \in F'$, $e' \in F$ and $F' \subseteq F$ it follows that $\{e, e'\} \subseteq F'$. Since we have assumed that F' is an MCCS, $\{\mathbf{t}(e), \mathbf{t}(e')\} \in \mathbf{R}$ should imply that $\{e, e'\} \in \mathcal{R}$. Thus, a contradiction has been reached. ■

Proposition 2: For any $F \in \mathcal{C}$ it holds that $Aug(F) \subseteq \alpha(F)$.

Proof: Since $\forall e \in Aug(F)$ it holds that $F \cup \{e\}$ is a CCS, it follows that $\forall e \in Aug(F)$ there exists an $F' \supseteq (F \cup \{e\}) \supseteq F$, $F' \in \mathcal{M}$. Since $\alpha(F)$ is defined as the union of all $F' \supseteq F$ for which $F' \in \mathcal{M}$, it follows that $Aug(F) \subseteq \alpha(F)$. ■

Proposition 3: For any $F \in \mathcal{C}$ such that $\mathbf{t}(Aug(F)) = \mathbf{t}(Comp(F))$ it holds that $Aug(F) = \alpha(F) = Comp(F)$.

Proof: From Propositions 1 and 2 we know that $Aug(F) \subseteq \alpha(F) \subseteq Comp(F)$. Thus to show equality of these three sets it suffices to show that $Aug(F) = Comp(F)$.

Assume that there exists an entity $e \in Comp(F)$ with $e \notin Aug(F)$ and $\mathbf{t}(Aug(F)) = \mathbf{t}(Comp(F))$. This means that $F \cup \{e\}$ is complete but not connected. Since F is connected, the only way for $F \cup \{e\}$ to be not connected is that $\forall e' \in F$ it holds that $\{e, e'\} \notin \mathcal{R}$. Combined with completeness of $F \cup \{e\}$, this means that $\nexists t \in \mathbf{t}(F)$ such that $\{t, \mathbf{t}(e)\} \in \mathbf{R}$. Indeed, if there was an $e' \in F$ such that $\{\mathbf{t}(e'), \mathbf{t}(e)\} \in \mathbf{R}$, then completeness would require that $\{e, e'\} \in \mathcal{R}$.

Since $\mathbf{t}(Aug(F)) = \mathbf{t}(Comp(F))$, there must be an entity $e'' \in Aug(F)$ with $\mathbf{t}(e'') = \mathbf{t}(e)$, such that e'' cannot be connected to any entity from F. This leads to a contradiction, as $F \cup e''$ can then not be connected and thus e'' cannot be an element of $Aug(F)$.

■

III. Algorithm

As shown in Definition 5, an α-CCS is defined as the union of all MCCSs that are supersets of a closed CCS. A naive way to enumerate all α-CCSs would thus be to enumerate all closed CCSs first, to then create all combinations of MCCSs that overlap with any given closed CCS. Clearly, such an algorithm would be prohibitively expensive though.

Instead, the algorithm we propose exploits the fact that RMiner, the algorithm to mine all MCCSs [15] traverses the set of all closed CCSs starting with the empty one and recursively expanding them until an MCCS is reached. From the associativity of the set union operation, it is clear that the α-CCS corresponding to a given non-maximal closed CCS is equal to the union of the α-CCSs corresponding to all direct expansions of that closed CCS. Thus, the RMiner recursive exploration can directly be used to compute all α-CCSs after making essentially one modification: each recursive call on a closed CCS F that yields a new closed CCS F' should now return the corresponding α-CCS $\alpha(F')$. This recursion ends as soon as F' is an MCCS, as then $\alpha(F') = F'$.

Proposition 3 allows a further optimisation: as soon as $\mathbf{t}(Aug(F)) = \mathbf{t}(Comp(F))$, it is no longer needed to compute the α-CCS corresponding to F by recurring, as then $\alpha(F) = Aug(F)$ which is already required and thus computed by RMiner. Note that RMiner enumerates all closed CCSs with polynomial delay [15], and this modification maintains this property for α-CCSs.

Below we explain this in greater detail by first recalling some details of RMiner, and then presenting the A-RMiner algorithm for enumerating all α-CCSs.

A. RMiner

RMiner it is an algorithm that lists all MCCSs by enumerating all closed CCSs, i.e., all CCSs that are fixpoints of the closure operator g as defined in Def. 4. It is an instantiation of the fixpoint listing algorithmic framework of Boley et al. [1].

More specifically, RMiner (shown in Algorithm 1) is a divide and conquer algorithm that works as follows. At every sub-call it chooses one element $e \in Aug(F) \setminus (F \cup B)$ (line 1), creates the closed set which corresponds to $g(F \cup \{e\})$ (line 2) and splits the search space in two branches, one producing solutions (supersets of F) containing e (line 7) and another producing solutions (supersets of F) not containing e (line 10). The set B contains all elements $e \in E$ that have been considered as extensions to subsets of F and has two functionalities. One is to ensure the split of the search space and the other one is to make sure that the same solution is not produced more than once, by pruning the branches that correspond to closed sets that have a non-empty intersection with it (line 3). A closed CCS is listed when it is maximal (lines 4 and 5).

Algorithm 1 RMiner: List all MCCSs

Main()

1: **if** $\rho(\emptyset) = Aug(\emptyset)$ **then**
2: Output $g(\emptyset)$
3: **end if**
4: RMiner$(g(\emptyset), \emptyset)$

RMiner(F, B)

1: Select $e \in Aug(F) \setminus (F \cup B)$
2: $F' = g(F \cup \{e\})$
3: **if** $F' \cap B = \emptyset$ **then**
4: **if** $F' = Aug(F')$ **then**
5: Output F'
6: **else**
7: RMiner(F', B)
8: **end if**
9: **end if**
10: RMiner(F, $B \cup \{e\}$)

B. Approximate RMiner

To describe Approximate RMiner (A-RMiner), recall that the α-CCS $\alpha(F)$ of a closed CCS F is defined as the union of all MCCSs that are supersets of F. These MCCSs correspond to the leaves of the part of the search space explored by RMiner that has F at its root. This is better depicted with an example.

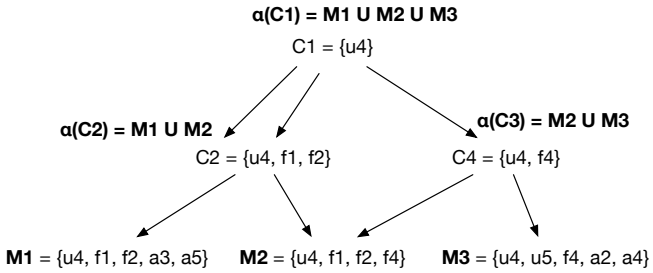

Fig. 2. Search space of closed CCSs corresponding to the toy data example of Fig. 1. It only shows part of the search space that has the entity u4 as root.

Example: Figure 2 shows part of the search space of closed CCSs corresponding to the toy data example of Fig. 1 and having the entity u4 as root. The MCCSs correspond to the leaves of the search tree. For every closed CCS that is not an MCCS we show the corresponding α-CCS as well. MCCSs are also themselves trivially α-CCS.

Using the associative property of the union we can employ the depth-first nature of RMiner to produce all α-CCSs incrementally. This can be done by outputting the union of all superset MCCSs of the every closed CCS, starting from the bottom of the search tree, and propagating it up one level.

As we saw in the previous subsection, RMiner uses the set B to prune a subtree of the search space when it is rooted in a closed CCS that has already been generated. This way it avoids creating duplicate solutions. In the case of mining α-CCSs, while we do not want to create duplicate solutions we cannot simply prune branches of the search space as this would mean that we loose some of the superset MCCSs of the current closed CCS which are used to create the α-CCSs above it. We show this with an example.

Example: In the search space of Fig. 2, C2 and M2 are produced twice. Using the fixpoint listing framework they would be pruned the second time they are produced. However, pruning M2 would result in an incorrect α-CCS, $\alpha(C3)$.

In order to avoid creating duplicate α-CCSs we make use of the fact that when for $F \in \mathcal{C}$ it holds that $F \cap B \neq \emptyset$ then $\forall F' \supseteq F$, $F' \in \mathcal{M}$ it holds that $(F' \cap B) \neq \emptyset$. Therefore when for $F \in \mathcal{C}$ it holds that $F \cap B \neq \emptyset$, all the α-CCSs that correspond to supersets of F have already been created in other branches. As a result there is no need to explore the search space under F.

However, for every $F \in \mathcal{C}$ for which $F \cap B \neq \emptyset$ we still need to know α(F) in order to propagate it up and build the α-CCSs that correspond to $F' \subset F$, $F' \in \mathcal{C}$. The straightforward approach to deal with this problem, would be to store all previously produced α-CCSs indexed by the their corresponding closed CCS. However this approach would require memory that is exponential to the input size. We propose an efficient way to deal with this problem, which is based on Proposition 3. Using this proposition we only need to store the α-CCSs corresponding to closed CCSs F for which $\mathbf{t}(Aug(F)) \neq \mathbf{t}(Comp(F))$.

Algorithm 2 shows A-RMiner. M is the α-CCS that is produced from every recursive call and N is a partial union corresponding to one branch of the search tree. P is the data structure which stores the α-CCS corresponding to a closed CCS when needed.

Algorithm 2 A-RMiner

Main()

1: Output LIST$(g(\emptyset), \emptyset)$

LIST(F, B)

1: $N = \emptyset$, $M = F$
2: **for** $e \in Aug(F) \setminus (F)$ **do**
3: $F' = g(F \cup \{e\})$
4: **if** $F' \cap B = \emptyset$ **then**
5: $N =$LIST(F', B)
6: Output N
7: **if** $\mathbf{t}(Aug(F')) \neq \mathbf{t}(Aug(F))$ **then**
8: $P[F'] = N$
9: **end if**
10: **else**
11: **if** $\mathbf{t}(Aug(F')) \neq \mathbf{t}(Aug(F))$ **then**
12: $N = P[F']$
13: **else**
14: $N = Aug(F')$
15: **end if**
16: **end if**
17: $M = M \cup N$
18: $B = B \cup \{e\}$
19: **end for**
20: **return** M

C. A-RMiner with additional constraints

As with all pattern mining algorithms, the overall time complexity of A-RMiner depends on the number of closed CCSs. In its general form, A-RMiner produces patterns containing one relationship as well as patterns containing more

than one relationships. As a way to further improve its time performance, we give the option to focus on truly relational α-CCSs such that the closed CCS in which they overlap contains more than one relationships. We do this by defining an additional constraint on the closed CCSs. This is a constraint which we have defined for the case of RMiner [15]. Here we give its definition and show how it is applicable for the case of A-RMiner.

More specifically we define a constraint c on the minimum number of entities per entity type in a closed CCS, and we refer to it as *minimum coverage constraint*. Formally:

Definition 6: For $F \in \mathcal{C}$, and for all $i \in \mathbf{t}(E)$ of a relational database $\mathbb{D} = (E, \mathbf{t}, \mathcal{R}, \mathbf{R})$, we fix a number $c_i \in \mathbb{N}$ and define

$$c(F) = \begin{cases} 1, & \text{if } \forall i \in \mathbf{t}(E), |F \cap E_i| \geq c_i \\ 0, & \text{otherwise} \end{cases} .$$

This constraint cannot directly be used for pruning, because it is not anti-monotone. To resolve this, we define $\bar{c}$ as follows.

Definition 7: For $F \in \mathcal{F}_{\mathbb{D}}$, and for all $i \in \mathbf{t}(E)$ of a relational database $\mathbb{D} = (E, \mathbf{t}, \mathcal{R}, \mathbf{R})$, we fix a number $c_i \in \mathbb{N}$ and define

$$\bar{c}(F) = \begin{cases} 1, & \text{if } \forall i \in \mathbf{t}(E), |(Comp(F) \setminus B) \cap E_i| \geq c_i \\ 0, & \text{otherwise} \end{cases} .$$

The fact that $\bar{c}$ is an anti-monotone constraint follows from the anti-monotonicity of the set Comp, and the fact that the set B is bigger for larger closed CCSs [15]. This means that for $F \subseteq F'$, if $\bar{c}(F) = 0$ then $\bar{c}(F') = 0$. However, to be able to safely use the constraint $\bar{c}$ for pruning we need to ensure that we are not loosing any of the α-CCSs such that the closed CCS they correspond to satisfies the constraint c. In case $F \cap B = \emptyset$, it holds that $F \subseteq (Comp(F) \setminus B)$ because $F \subseteq Comp(F)$. Therefore when $F \cap B = \emptyset$ and $\bar{c}(F) = 0$ it holds that $c(F) = 0$. Taking into account the the anti-monotonicity of $\bar{c}$ as well, it is safe to prune in this case. In case $F \cap B \neq \emptyset$, the search space under F has already been explored and it is going to be pruned by the algorithm anyway.

To use this constraint in Algorithm 2 line 4 is modified as follows:

$$4 : \textbf{if } F' \cap B = \emptyset \wedge \bar{c}(F') = 1 \textbf{ then}$$

IV. Interestingness

To compute the interestingness of α-CCSs, the general framework from [5], [3] provides a justification for the quantification of the interestingness of a pattern as the ratio of its information content and its description length. Let us discuss these two components in turn.

A. Information content

The information content of a pattern is computed as the negative log probability under a probability that represents the belief state of the user. The smaller the probability the pattern has under this so-called *background distribution*, the more information the pattern conveys to the user. This amount of subjective information is what the *information content* aims to quantify (see [5], [3] for details).

Thus, to compute the information content, we need to settle on a type of prior beliefs, and derive the background distribution for such prior beliefs. Here we follow the same approach as in [13], [14], but it should be noted that other prior beliefs and resulting background distributions may be more appropriate, depending on the situation.

To recall, in these papers it was assumed that the user holds prior beliefs about the number of relationship instances that each entity participates in. I.e., considering the relationship type $\{i, j\}$, it is assumed that the user has the correct expectation about the value of $d^e_{\{i,j\}} \triangleq \sum_{e' \in E_j} I(\{e, e'\} \in \mathcal{R}_{\{i,j\}})$ for any $e \in E_i$ (where I is the indicator function), as well as about the value of $d^{e'}_{\{i,j\}} \triangleq \sum_{e \in E_i} I(\{e, e'\} \in \mathcal{R}_{\{i,j\}})$ for any $e' \in E_j$.

Given such prior beliefs, the background distribution is equal to a product of distributions for each relationship type separately:

$$P(\mathbb{D}) \quad = \quad \prod_{\{i,j\} \in \mathbf{R}} P_{\{i,j\}}(\mathcal{R}_{\{i,j\}}).$$

The distributions of the individual relations also factorise as a product distribution over all possible pairs of entities within the relationship type:

$$P_{\{i,j\}}(\mathcal{R}_{\{i,j\}}) \quad = \quad \prod_{e \in E_i, e' \in E_j} P^{e,e'}_{\{i,j\}}(I(\{e, e'\} \in \mathcal{R}_{\{i,j\}})),$$

where I is the indicator function and $P^{e,e'}_{\{i,j\}}$ is a Bernoulli distribution defined by:

$$\begin{aligned} P^{e,e'}_{\{i,j\}}(1) &= \frac{\exp\left(\lambda^e_{\{i,j\}} + \lambda^{e'}_{\{i,j\}}\right)}{1 + \exp\left(\lambda^e_{\{i,j\}} + \lambda^{e'}_{\{i,j\}}\right)}, \\ P^{e,e'}_{\{i,j\}}(0) &= 1 - P^{e,e'}_{\{i,j\}}(1). \end{aligned}$$

Here, the parameters $\lambda^e_{\{i,j\}}$ and $\lambda^{e'}_{\{i,j\}}$ should be such that the expected degrees are equal to the values $d^e_{\{i,j\}}$ and $d^{e'}_{\{i,j\}}$ the user expects them to have. This can be done for each relationship type separately, by solving a highly scalable convex optimisation problem as detailed in [4].

Using this background distribution, it is then trivial to compute the probability that a given α-CCSs is present in the data. Indeed, this can be done by summing over all relationship types involved in the pattern, and all pairs of entities involved in that relationship type, the negative log probability that the pair of entities is related or not (whichever is the case) in the α-CCS. Formally:

$$\begin{aligned} &\text{InformationContent}(F) = \\ &- \sum_{\{i,j\} \in \mathbf{R}} \sum_{\substack{e \in F \cap E_i \\ e' \in F \cap E_j}} \log\left(P^{e,e'}_{\{i,j\}}\left(I\left(\{e, e'\} \in \mathcal{R}_{\{i,j\}}\right)\right)\right). \end{aligned}$$

B. Description length

The description length is a quantification of how much effort the user needs to expend to assimilate the pattern, which we contend can be approximated by means of the length of the description of the pattern in the language (i.e. encoding) the user will find easy to use.

We define the description length of a pattern in two parts. The first part is the number of bits required to describe that an entity is or is not part of the pattern and the second part is the number of bits required to describe which of the possible relationship instances in the pattern are missing. This is analogous to [10], which formalises the description length of noisy tiles.

Formally the description length of an α-CCS F is given by the following formula:

$$\text{DescriptionLength}(F) = -\sum_{e \in F} \log(p) - \sum_{e \notin F} \log(1-p) + \log\left(\binom{n_F}{k_F}\right),$$

where p is a parameter for which the overall density of the database appears to be a good value (i.e. the cardinality of $\mathcal{R}$ divided by the maximal cardinality it could have given E and $\mathbf{R}$), and where

$$n_F = \sum_{\{i,j\} \in \mathbf{R}} \mid F \cap E_i \mid \times \mid F \cap E_j \mid,$$

$$k_F = \sum_{\{i,j\} \in \mathbf{R}} \sum_{\substack{e \in F \cap E_i \\ e' \in F \cap E_j}} I(\{e, e'\} \notin \mathcal{R}_{\{i,j\}}).$$

The first two logarithmic terms account for the cost of describing which entities are and which are not present in F. The logarithmic binomial term accounts for the cost of describing which k_F of the n_F possible relations between these entities are not present in the database. (An extra small term accounting for the description of the number k_F is omitted for brevity).

V. Experiments

In this section we present experiments that show the usefulness of the pattern syntax, its resilience to noise and the performance of the algorithm.

A. Useful Patterns

In order to show the usefulness of our pattern syntax we run A-RMiner on the MovieLens dataset [9]. This dataset contains information about films as well as ratings of films. Ratings vary from 1 to 5 but we actually considered that a user likes a film if they have given rating between 4 and 5 and do not like a film if they have given less. The dataset we used contained four entity types: users, films, directors and genres; and 3 relationship types: one between users and films specifying whether a user liked a film, one between films and directors and one between films and genres.

In Fig. 3 we see a visualisation of the top ranked pattern when we required the closed CCSs to contain at least one director, six films, six users and one genre. This pattern contains a group of 37 users, 6 films, 1 director and 2 genres. All films are directed by the same director however not all of them are of both genres shown in the pattern and not all users in the pattern like all of the films. It is ranked high by the interestingness measure as it contains a lot of information (960 relationship instances) conveyed in a concise way (46 entities). Such a pattern could be used to identify groups of users that like similar films.

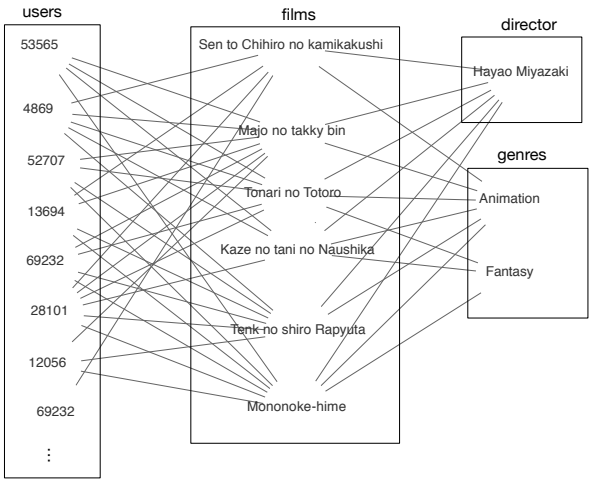

Fig. 3. Visualisation of the most interesting α-CCS pattern on the movieLens dataset when requiring the closed CCSs to contain at least one director, six films, six users and one genre. The pattern is represented as a k-partite graph where entity types correspond to different partitions and edges are drawn if there exists a relationship instance between the entities. The pattern actually contains 37 users most of which we omitted for the clarity of the visualisation.

B. Mining noise tolerant patterns

Here we present an experiment which shows the resilience of α-CCSs to noise. More specifically we created a random data set which contains three entity types and two relationship types. Every entity type contained 100 entities. Entities were connected with probability 0.01. We then embedded an MCCS in this dataset by randomly selecting 8 entities from every entity type and added all possible relationship instances. Finally we added noise to the data by XORing it with another random dataset in which entities were connected with probability (noise level) 0.01 and 0.1. We report the Jaccard distance between the top 10 most interesting α-CCSs and the MCCS that we embedded for different levels of noise.

The results are shown in Table I. At noise level 0, 100% of the embedded MCCS is retrieved as expected. It is retrieved at rank one and also part of it at other ranks as it contains a lot of information compared to the rest of the dataset which is otherwise random. But even when there is noise a very large percentage of it is retrieved at high ranks. This shows that the pattern syntax of α-CCSs is quite resilient to noise.

C. Scalability

Here we present a scalability experiment showing how A-Rminer scales (both in time and space) for different input sizes. We also compare the scalability of A-RMiner to that of RMiner. The datasets for this experiment were extracted from IMDB. We produced snapshots of it containing the films, genres and directors that correspond to 1, 2, 10, 40 and 100 years of cinema starting from the year 1910. The results of the experiment are shown in Table II. A-RMiner runs in time and space comparable to that of RMiner. As in the case of RMiner, the total time of A-RMiner seems to scale polynomially to the input size and space linearly to the input size.

TABLE II. SCALABILITY TESTING OF A-RMINER ON DATASETS OF INCREASING SIZE OF ENTITIES TAKEN FROM THE IMDB.

Constraints	Number of Entities	Number of α-CCSs	Time (sec)	Space (Mb)	Number of MCCSs	Time_RMiner (sec)	Space_RMiner (Mb)
(1,1,1)	3,291	945	0.42	5.9	491	0.36	5.2
	8,686	2,181	3.81	15.7	980	3.55	11.2
	51,203	18,498	260	103.6	7,621	229	107.6
	111,320	88,755	1,512.7	241	32,213	1,227.8	218
	514,323	46,8976	38,806	564	253,148	37,750	481
(2,2,2)	3,291	3	0.01	4.9	3	0.01	4.6
	8,686	24	0.1	10.2	23	0.1	10.2
	51,203	143	8.78	5.3	125	8.59	5.3
	111,320	570	50.5	114.4	420	49.3	118.1
	514,323	1,725	630.5	500	1,286	583.04	605

TABLE I. JACCARD DISTANCE (IN TERMS OF ENTITIES) BETWEEN THE EMBEDDED MCCS AND THE TOP 10 MOST INTERESTING PATTERNS FOR DIFFERENT NOISE LEVELS AND FOR DATA DENSITY 10^{-2}.

	Noise		
Rank	0	0.01	0.1
1	100%	96%	75%
2	96%	96%	75%
3	96%	100%	75%
4	96%	72%	79%
5	96%	92%	71%
6	63%	56%	71%
7	65%	60%	75%
8	0%	96%	75%
9	65%	57%	71%
10	0%	92%	71%

VI. CONCLUSION

We presented the pattern syntax of α-CCSs for mining approximate patterns in multi-relational data as well as a way to rank such patterns based on their subjective interestingness. To the best of our knowledge this is the first approximate pattern syntax for the case of multi-relational data. The definition we proposed is very intuitive and, as shown in the experiments, also useful and quite resilient to noise. Experiments using various constraints showed that the mining algorithm we proposed is as efficient as the one that mines exact patterns and scales polynomially to the input size.

REFERENCES

[1] M. Boley, T. Horvath, A. Poigné, and S. Wrobel. Listing closed sets of strongly accessible set systems with applications to data mining. *Theoretical Computer Science*, 411(3):691 – 700, 2010.

[2] H. Cheng, P. Yu, and J. Han. Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 839–844, 2006.

[3] T. De Bie. An information-theoretic framework for data mining. In *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD11)*, 2011.

[4] T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23(3):407–446, 2011.

[5] T. De Bie, K.-N. Kontonasios, and E. Spyropoulou. A framework for mining interesting pattern sets. *SIGKDD Explorations*, 2010.

[6] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[7] A. Goldberg. Finding a maximum density subgraph. Technical report, 1984.

[8] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 301–309, 2008.

[9] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 230–237, 1999.

[10] K.-N. Kontonasios and T. De Bie. An information-theoretic approach to finding informative noisy tiles in binary databases. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pages 153–164, 2010.

[11] J. Liu, S. Paulsen, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets from noisy data. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 721–724. IEEE, 2005.

[12] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[13] E. Spyropoulou and T. De Bie. Interesting multi-relational patterns. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2011.

[14] E. Spyropoulou, T. De Bie, and M. Boley. Mining interesting patterns in multi-relational data with n-ary relationships. In *Proceedings of the International Conference on Discovery Science (DS)*, pages 217–232, 2013.

[15] E. Spyropoulou, T. De Bie, and M. Boley. Mining interesting patterns in multi-relational data. *Data Min. Knowl. Discov.*, 2014.

[16] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[17] T. Uno. An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica*, 56(1):3–16, 2010.

[18] C. Yang, U. Fayyad, and P. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 194–203, 2001.